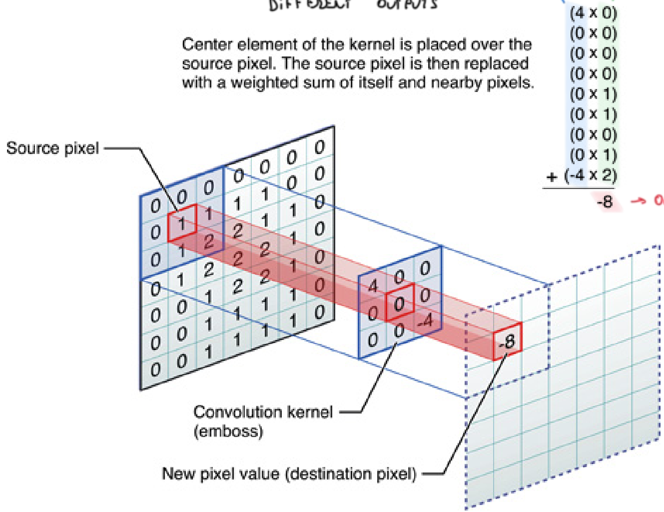
Convolutional Neural Networks

Filters and convolutions

SLIDING WINDOW MOVING OVER THE INPUT IN DIFFERENT POSITIONS IN ORDER TO OBTAIN DIFFERENT OUTPUTS ← KERNEL (PATTERN)

INPUT



→ OUTPUT

THEY REPRESENT A SORT OF MAPPING OF THE INPUT

Filters and convolutions

(KERNEL)

filter

0	1	0
1	–4	1
0	1	0

input

6	2	3	1	0
9	5	1	1	3
2	6	5	8	7
3	4	4	8	3

output

–2		

Filters and convolutions

filter

0	1	0
1	−4	1
0	1	0

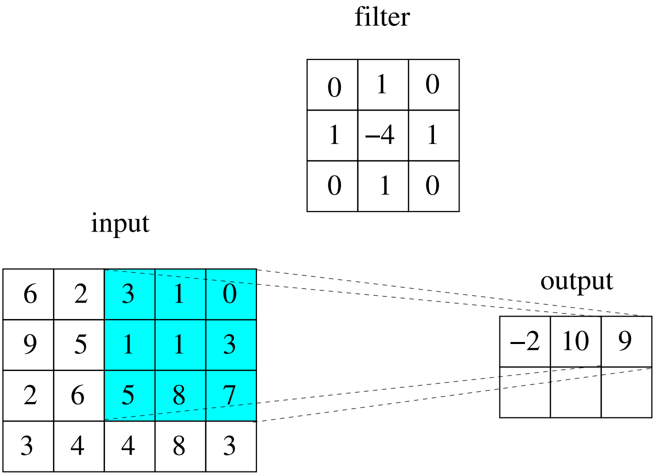
input

6	2	3	1	0
9	5	1	1	3
2	6	5	8	7
3	4	4	8	3

output

−2	10	

Filters and convolutions



Filters and convolutions

filter

0	1	0
1	−4	1
0	1	0

input

6	2	3	1	0
9	5	1	1	3
2	6	5	8	7
3	4	4	8	3

output

−2	10	9
−8		

Filters and convolutions

filter

0	1	0
1	−4	1
0	1	0

input

6	2	3	1	0
9	5	1	1	3
2	6	5	8	7
3	4	4	8	3

output

−2	10	9
−8	−1	

Filters and convolutions

filter

0	1	0
1	−4	1
0	1	0

input

6	2	3	1	0
9	5	1	1	3
2	6	5	8	7
3	4	4	8	3

output

−2	10	9
−8	−1	−11

Loose connectivity and shared weights

- the activation of a neuron is not influenced from all neurons of the previous layer, but only from a small subset of adjacent neurons: his receptive field
- every neuron works as a convolutional filter. Weights are shared: every neuron perform the **same trasformation** on **different areas** of its input
- with a cascade of convolutional filters intermixed with activation functions we get complex non-linear filters assembing local features of the image into a global structure.

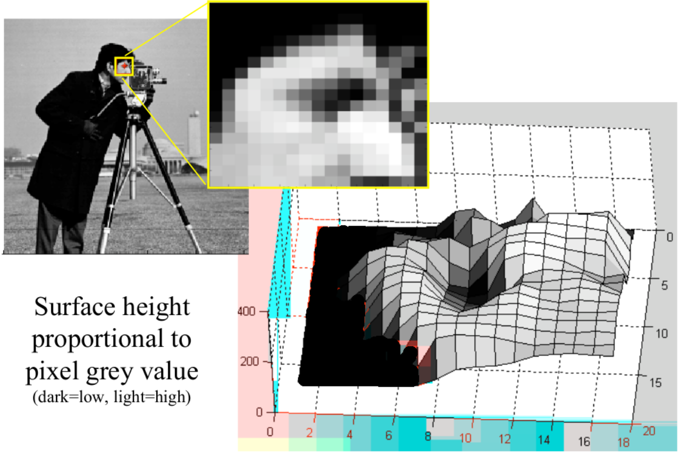
About the relevance of convolutions for image processing

Images are arrays

An image is coded as a numerical matrix (array)
grayscale (0-255) or rgb (triple 0-255)

$$\begin{bmatrix} 207 & 190 & 176 & 204 & 204 & 208 \\ 110 & 108 & 114 & 112 & 123 & 142 \\ 94 & 100 & 96 & 121 & 125 & 108 \\ 95 & 86 & 81 & 84 & 88 & 88 \\ 69 & 51 & 36 & 72 & 78 & 81 \\ 74 & 97 & 107 & 116 & 128 & 133 \end{bmatrix}$$

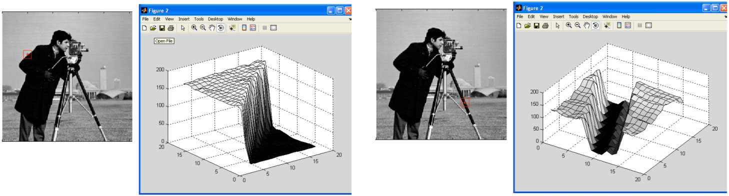
Images as surfaces



Surface height proportional to pixel grey value
(dark=low, light=high)

Interesting points

Edges, angles, ...: points where there is a discontinuity, i.e. a fast variation of the intensity



We measure variations of intensitites by means of derivatives and we can compute **discrete approximations** of derivatives convolving simple **linear filters**

Approximation of the derivative

finite central difference

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2)$$

Usually, $h = 1$ pixel; neglecting the constant $\frac{1}{2}$ we compute the derivative with the following filter:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Example

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$\longrightarrow$$

$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

$$\longrightarrow$$

Linear Filters

The derivative is an example of linear filter

Idea:
create new images where each pixel is a linear combination (defined by a kernel) of the adjacent pixels
The same trasformation is repeatedly applied centering the kernel on every pixel (convolution)

Examples:
- blurring (mean)
- gaussian smoothing
- edge detection
- sharpening
- embossing
- and many others ...

relevant properties

- the output is a linear tranformation of the input
- a shift of the input results in a shift of output
- linear filter can be combined

Demo

Vedere http://docs.gimp.org/2.8/en/plug-in-convmatrix.html

FOR EACH POINT, COMPUTES THE AVERAGE AMOG THE NEAR POINTS

blur $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

sharpen $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

FOR EACH POINT, INTENSIFIES ITS VALUE AND CONSIDER TE OPPOSIT OF THE NEAR ONES

THE MORE YOU PUT THERE AND THE MORE SHARPEN IT IS

↓

IT'S LIKE RE-APPLYING THE IDENTITY FILTER MORE AND MORE OVER AN EDGE DETECTOR

↓

ORIGINAL PICT + EDGES

Demo

edge-detect $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ emboss $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$

IN A SENSE WE ARE ADDING SHADOWS TO IMAGES $\downarrow$ MAKE A PHOTO SEEM 3D

Convolution, formally

Convolution is a mathematical operation transforming an input matrix by means of another matrix (kernel).

The operation can be generalized to the continous case (tranforming a function via another function)

In the binary case, given a function $f(x, y)$ and a kernel $k(x, y)$ the convolution $f * k$ of f and k is defined as follow

$$f(x,y) * k(x,y) = \begin{cases} \int_u \int_v f(x-u, y-v) \cdot k(u,v) & \textit{continuous} \\ \sum_u \sum_v f(x-u, y-v) \cdot k(u,v) & \textit{discrete} \end{cases}$$

Convolution is symmetric, associative, and distributive.

Convolution and correlation

Having a kernel in the interval $[-M, M]$,

$$(f * k)(x) = \sum_{m=-M}^{M} f(x-m) \cdot k(m)$$

Observe tat $k(-M)$ is the multiplicative factor for $f(x+M)$, that is, the kernel must be **flipped**, before taking products.

If you do not flip the kernel, you get a different transformation known as cross-correlation.

- not relevant if the kernel is symmetric
- not so relevant for Neural Nets, since weights are generated by the machine

Back to CNNs

Inferring features

Usual idea:

instead of using pre-defined filters, let the net learn its own filters.

Particularly important in deep architectures.

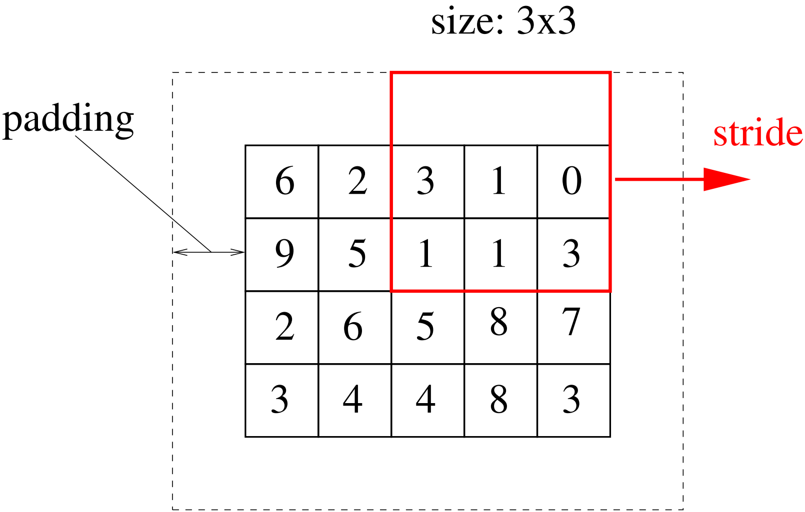
Spatial structure

A convolutional layer is defined by the following parameters

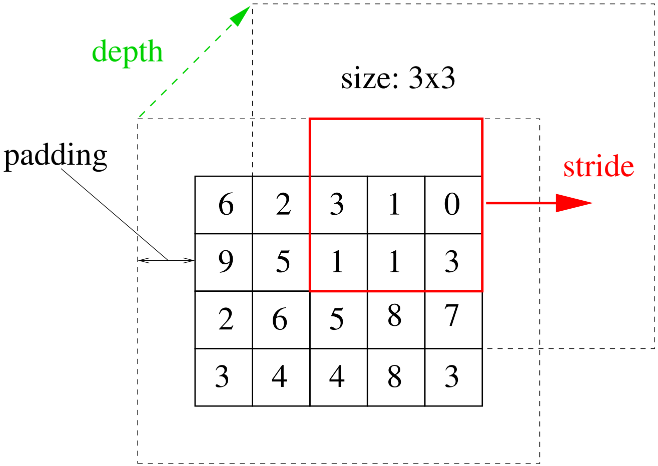
- **kernel size**: the dimension of the linear filter.
- **stride**: movement of the linear filter. With a low stride (e.g. unitary) receptive fields largely overlap. With a higher stride, we have less overlap and the dimension of the output get smaller (lower sampling rate).

  INVERSALLY PROPORTIONED TO OVERLAP
- **padding** Artificial enlargement of the input to allow the application of filters on borders.
- **depth**: number of different kernels that we wish to syntesize. Each kernel will produce a different feature map with a smae spatial dimension.

Layers configuration params



Layers configuration params



Dimension of the output

The spatial dimension of each output feature map depends form the spatial dimension of the input, the padding, and the stride. Along each axes the dimension of the output is given by the formula

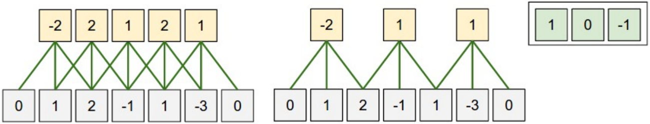
$$\frac{W + P - K}{S} + 1$$

where:

W = dimension of the input
P = padding
K = Kernel size
S = Stride

Example (unidimensional)

The width of the input (gray) is W=7.
The kernel has dimension K=3 with fixed weights $[1, 0, -1]$
Padding is zero



In the first case, the stride is S=1. We get $(W - K)/S + 1 = 5$ output values.
In the second case, the stride is S=2. We get $(W - K)/S + 1 = 3$ output values.

Example 2D

INPUT $[32 \times 32 \times 3]$ color image of 32×32 pixels. The three channels R G B define the input depth

CONV layer. Suppose we wish to compute 12 filters with kernels 6×6, stride 2 in both directions, and zero padding. Since $(32-6)/2+1 = 14$ the output dimension will be $[14 \times 14 \times 12]$

RELU layer. Adding an activation layer the output dimension does not change

Padding modes

Usually, there are two main "modes" for padding:

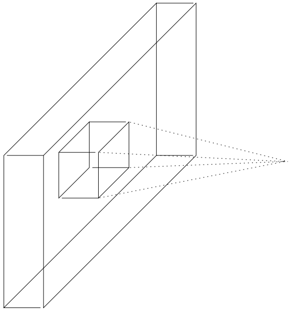
- **valid** no padding is applied
- **same** you add a minimal padding enabling the kernel to be applied an integer number of times

Important remark

Unless stated differently (e.g. in separable convolutions), a filter operates on all input channels in parallel.

So, if the input layer has depth D, and the kernel size is NxM, the actual dimension of the filter will be

NxMxD



The convolution kernel is tasked with simultaneously mapping cross-channel correlations and spatial correlations

Pooling

In deep convolutional networks, it is common practice to alternate convolutional layers with pooling layers, where each neuron simply takes the mean or maximal value in its receptive field.

This has a double advantage:

- it reduces the dimension of the output
- it gives some tolerance to translations

Max Pooling example

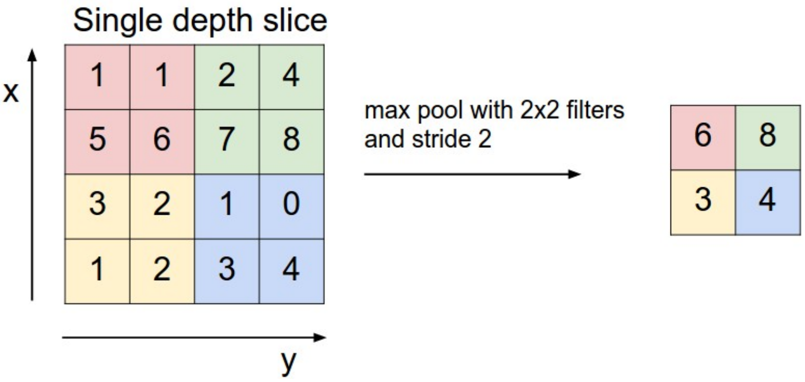
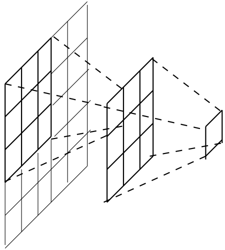


Immagine tratta da
http://cs231n.github.io/convolutional-networks/

Receptive field

The receptive field of a (deep, hidden) neuron is the dimension of the input region influencing it.

It is equal to the dimension of an input image producing (without padding) an output with dimension 1.



A neuron cannot see anything outside its receptive field!

Parameters and Flops

Parameters

The number of parameters of a **Dense layer** depends from the spatial dimension S_{in} of the input and the spatial dimension S_{out} of the output:

$$\text{params} = S_{in} \times S_{out} + S_{out}$$

The number of parameters of a **Convolutional layer** depends from the spatial dimension of the kernel (say, $K_1 \times K_2$) times the input-depth C_{in} (this product is the size of each kernel), times the output-depth (this is the number of different kernels that are syntesized)

$$\text{params} = K_1 \times K_2 \times C_{in} \times C_{out} + C_{out}$$

(each kernel potentially has its own bias)

Floating Point Operations (Flops)

The number of Flops required to apply a **Dense layer** it proportional to its parameters:

$$\text{flops} \sim S_{in} \times S_{out}$$

The number of flops required to apply a **Convolutional layers** is proportional to the kernel parameters, multiplied by the number of applications of each kernel, that is equal to the output spatial dimensions

$$\text{flops} \sim K_1 \times K_2 \times C_{in} \times C_{out} \times W_{out} \times H_{out}$$

Flops as a Cost measure

Flops are not a good cost measure:

On GPU-like arichitectures, convolutions are easily and cheaply paralllelized along the spatial dimension (same for dense layers along batchsize).

The right measure is between params and flops, depending on the architecture, see:
Dissecting FLOPs along input dimensions for GreenAI cost estimations